
Opening the Networks with Parlay/OSA: Standards and Aspects Behind the APIs

Ard-Jan Moerdijk and Lucas Klostermann, Ericsson Eurolab Netherlands

Abstract

Recently there has been an enormous increase in efforts to open up telecom networks for application development. In opening up the network, new business models emerge where applications can be developed and provided by enterprises outside the traditional network operator domain. This new business model, combined with the fact that applications can be built using standardized APIs with off-the-shelf IT technology and tools, will result in new innovative applications that will hit the market with drastically reduced development cycles. This article provides an overview of the Parlay/OSA initiatives concerning the specification of a set of open standardized APIs. Furthermore, the article outlines some architectural aspects that are implied and implicitly contained in the Parlay/OSA specifications. These are critical for understanding the implications when allowing access to the network via Parlay/OSA. The article then outlines different types of applications that can be built using Parlay/OSA. Finally, we conclude with an analysis of the future of Parlay/OSA.

In today's telecommunications networks, applications and services are a part of the network operator's domain and are primarily built using intelligent network (IN) technology [1]. This approach is well suited to simple mass market and carrier-class applications. However, with the introduction of mobility and IP in the networks, easy creation and rapid deployment of innovative applications become challenges beyond the capability of IN.

In the past initiatives like Telecommunications Information Networking Architecture (TINA) [2] tried to address this challenge, but the results have so far largely remained at the research level. Nonetheless, inspired by the TINA work, different groups like Parlay [3] and JAIN [4, 5] continued with efforts to develop application programming interfaces (APIs) that allow applications access to network functionality.

The main starting points used to address the challenge for new innovative applications are that:

- The APIs should address a broad scope of network functionality. In principle, all functionality inside the telecom network should be accessible via the APIs. This allows applications to combine different network capabilities using a uniform programming paradigm.
- The APIs should be based on open information technology (IT) in order to be accessible to a wide developer community.
- The APIs should abstract from existing network protocols and allow applications to be developed without telecom expertise. With the right abstraction, this also allows the APIs to work in different network environments, such as second-generation (2G) fixed, mobile, or 3G networks. This has the benefit that an operator with a 2G network is able to start with new innovative applications and migrate the network later, while the applications remain largely untouched.
- The APIs should support new business models. This allows a network operator to focus on mass market applications while extending its application portfolio, by offering niche market applications provided by third-party application providers.

All of these aspects are usually referred to as the opening up of the network, and will make telecom functionality accessible to a large developer community.

The APIs on which this article focuses were originally defined within the Parlay Group [3] and are standardized by the 3G Partnership Project (3GPP) [6] and European Telecommunications Standards Institute (ETSI) [7].

In the following we give a short overview of the history and current status of the Parlay/OSA API work. Next we look at the implied architecture of Open Service Access (OSA), where we outline some concepts that are key to OSA, but also somewhat implicit in the specifications. Following on from here, the application side of the interface is discussed, and some typical example applications are considered. We conclude with our view of the future evolution of Parlay/OSA APIs.

Parlay/OSA API Development and Standards Bodies Involved in Parlay/OSA API Development

Here we give a brief explanation of the different bodies involved in the production of the Parlay/OSA API, and how the API is developed and maintained.

In 1998, the Parlay group was established as an industry forum with 5 initial member companies. The first release of Parlay was completed within one year. As work continued on Parlay 2, new member companies joined the group. As the work on Parlay 2 commenced, the first release of the Parlay API was taken into ETSI and the International Telecommunication Union — Telecommunication Standardization Sector (ITU-T) with the intention of internationally standardizing the work. Around the same time members of the Parlay group initiated work within 3GPP on a similar API that could be used to provide access to applications on top of a Universal Mobile Telecommunications System (UMTS) network. As there was now a risk of developing three different APIs for the same purpose, it was deemed appropriate to combine the

groups into one joint working group in the context of what is called Open Service Access. In addition to the relevant Parlay APIs, new ones were defined and fed back into Parlay.

In order to ease specification work, with most of the individual member companies being members of all three bodies, it was decided not to have separate meetings. There were of course legal hurdles to overcome with the amalgamation of the three groups. As ETSI was an organizational partner with 3GPP, only an agreement between Parlay and ETSI was needed to make joint work formally possible. Nowadays all the work on the core set of the Parlay/OSA APIs is done in collocated meetings, and all the participants together are usually referred to as the Joint API Group or Joint Working Group (JWG). This group has the responsibility to further develop and maintain the Parlay/OSA specifications. In addition to the joint work, there is an informal collaboration with the JAIN community [4], mainly for the purpose of developing Java versions of the APIs. Furthermore, there are a number of Parlay Working Groups that take responsibility for developing specific new APIs. This provides an opportunity to try out new ideas and allows work to be done on a specific task with a focused incubator group. When the work of such a group is successful, the Joint API Group adopts the additional APIs.

At the moment of writing 3GPP has produced OSA Release 4 and 5 specifications, while ETSI and Parlay have completed corresponding common specification sets called ETSI ES 201 915/Parlay 3.2 and ETSI ES 202 915/Parlay 4.0. Furthermore, 3GPP2 has decided to also adopt the Parlay/OSA APIs.

As a final remark, we would like to point out that when different bodies have a common goal, we learned that it is very beneficial to work together from the start and adopt one process. Furthermore, we also learned that when one has succeeded in having the work done within a single group, it helps the specification work when requirements and release schedules are also aligned between the different bodies.

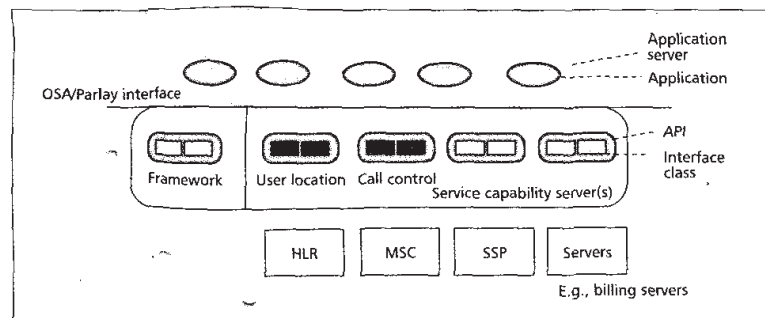
Parlay/OSA API Specification Overview

For the reader of the different specifications, the concepts used may still be a bit confusing due to the use of different terminology. In early Parlay specifications, the term *service* is used for the collection of interfaces that provide a specific function, while 3GPP and ETSI use the notation service capability feature (SCF) for the same concept. In the 3GPP text the term *service* usually means an end-user service.

In the remainder of this article the term *application* will be used for end-user service and the term *API* instead of service or SCF because API is a broadly known term within a wide developer community.

Parlay v. 4.0, 3GPP OSA Release 5, and ETSI ES 202 915 comprise the parts described in Table 1 [8–10].

All the parts are structured in the same way and contain use cases in the format of sequence diagrams, class diagrams, interface definitions, state diagrams, service properties, data definitions, and API realizations for the different technologies (OMG IDL, WSDL, and Java). For more details about the APIs themselves or the way the specifications are structured, the reader is referred to the introduction or the opening paragraphs of the different parts themselves. The specifications can be obtained from the Web sites of the three bodies [8, 9, 10]. In addition, [11] contains more of these details.



■ Figure 1. Overview of the logical entities involved in Parlay/OSA.

Behind the APIs: Architectural Aspects

In this section aspects of the OSA architecture are described. First of all, the logical entities are explained. Second, the aspects related to deployment in the network are described. Finally, some key architectural aspects such as security and scalability are outlined. As this article is intended as an overview, a comprehensive and exhaustive look at all architectural aspects is not possible.

The Logical Architecture

Figure 1 shows the logical architecture behind OSA. The logical entities to be distinguished are applications, application servers, service capability servers (SCSs), the Parlay/OSA Framework, and core network elements.

Applications use the capabilities defined in Parlay/OSA and offered through the Parlay/OSA APIs. Applications can be deployed on application servers, which can be any standard IT platform, and the capabilities are offered by SCSs: the SCS implements the server side and the application implements the client side of the API. Communication between the application and the SCS is done using standard IT middleware infrastructure (e.g., Common Object Request Broker Architecture, CORBA). SCSs are thus logical entities that implement the API and potentially interact with core network elements. These may include a home location registry (HLR), mobile switching center (MSC), service switching point, and so on. As such, an SCS serves as a proxy or gateway to the core network. Application servers could be in either the same business domain as the SCSs or in a different domain.

An SCS may implement multiple APIs, although one usually talks about a specific SCS per API, such as the call control SCS. Furthermore, since an SCS is a logical entity, it does not need to be implemented as a separate box. For instance, the content-based charging API might very well be offered directly by the charging and billing server itself. This is further detailed later.

The Parlay/OSA Framework, which is also shown in Fig. 1, offers the Framework capabilities to applications. This entity is essential in making it possible to go beyond traditional IN when it comes to openness, discovery, and integration of new features, as described below. The following section explains this crucial part of Parlay/OSA on a high level.

The Role of the Parlay/OSA Framework — The Parlay/OSA Framework builds on ideas from TINA [2] and provides controlled access to the APIs. In combination with distributed technology, the FW supports flexibility in application location and business scenarios. Furthermore, the Framework allows multivendorship and even the inclusion of nonstandardized APIs, which is crucial for innovation and service differentiation.

The Parlay/OSA Framework consists of a family of interfaces with the core part consisting of:

- Trust and security management: authentication of domains

Part #	API	High-level description	Comments
1	General	Contains the introduction, methodology, and design paradigms used.	
2	Common data	This part contains the common data definitions (i.e., generic data types) that are used by multiple APIs.	
3	Framework	Contains the infrastructure APIs, such as authentication between two domains, discovery of supported non framework APIs, registration of non-Framework APIs, fault management APIs, etc. The role of the Framework is considered in more detail later.	
4	Call control	The call control API family: generic call control, multiparty call control, multimedia call/session control, and Conference call control APIs. The capabilities supported by these APIs range from setting up basic calls to manipulating multimedia conference calls.	Conference call control not (yet) part of 3GPP OSA
5	User interaction	The specification for the user interaction API that allows applications to obtain information from the end-user, play announcements, send short messages, etc.	
6	Mobility	This part contains the APIs that allow applications to obtain information about the status and location of a person.	
7	Terminal capabilities	This API supports obtaining the capabilities of an end-user terminal.	
8	Data session control	The API in this part allows applications to influence data sessions initiated by a mobile terminal.	
9	Generic messaging	The API in this part allows access to mailboxes, and sending and retrieving messages.	Currently not part of 3GPP OSA
10	Connectivity management	This part specifies the API that allows applications to control or influence the QoS between endpoints in the network.	Currently not part of 3GPP OSA
11	Account management	The API in this part allows applications to query account balances and retrieve transaction histories.	
12	Content-based charging	This part contains the API that allows applications to charge parties for the services offered (e.g., content delivered).	
13	Policy management	This API allows setting up policies and registering for policy related events.	
14	Presence and availability management	This API allows applications to obtain and set information about a user's presence and availability.	

■ Table 1. Parlay/OSA APIs and their functional areas.

- Registration: registration of new SCSs to the Framework
 - Service life cycle manager: creation of a new instance of the API implementation
 - Service discovery: discovery of APIs provided in the target network
- In addition there are also Framework interfaces for:
- Integrity management: load balancing, fault management, heartbeat
 - Event notification: notifications for specific events (e.g., registration of new SCS)
 - Contract management: management of contracts between different domains (e.g., between application provider and network operator)

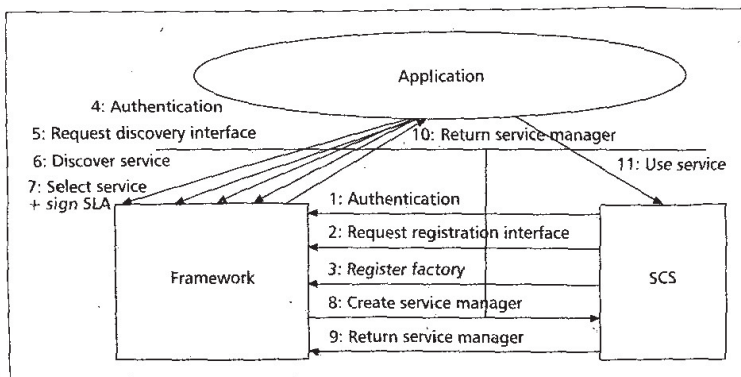
To explain the core part of the Framework, it is beneficial to look at a complete process where a new SCS is installed and an application starts using the capabilities offered by this SCS. The scenario is outlined in Fig. 2. Three different stages can then be distinguished:

Registration: Adding a new SCS: First, the SCS contacts the Framework and makes a request for the registration interface. These are steps 1 and 2 in Fig. 2. Next, the new SCS uses the registration interface to publish its type (e.g., user location) and its capabilities. Also, the SCS will supply the Framework with its own reference, provided by the service life cycle manager. This is step 3 in Fig. 2. At this point in time the Framework and SCS recognize each other.

Setup of service agreement: Establishing the conditions under which an application is allowed to use the APIs: From this point on, the Framework operator is able to enter the conditions under which an application is allowed to use the SCS, using as a basis the capabilities that the SCS supplied during registration. All of this information is captured in what is called a service agreement, and the information is entered via a management system.

Runtime communication establishment: The final step occurs when an application contacts the Framework and uses the discovery interface to find out what APIs are supported (steps 4–6). The application will first request the Framework via the discovery interface, to return all implementations of a certain API. The Framework then checks the service agreement to find out whether the application is indeed allowed to use this API. If this is the case, it will return a list of implementations. Next, the application can select one from the list (step 7 in Fig. 2). After this the Framework will request the specific SCS to create an instance that will be uniquely for this application (step 8). The SCS will now create the instance and return a reference to the Framework (step 9). The Framework will then return this to the application (step 10). From this moment on, the application is able to use the SCS.

Steps 6–10 are to be repeated for each API that the application wants to use.



■ Figure 2. Service registration and discovery: steps 1–3 are part of the registration step; steps 4–11 are part of the runtime communication establishment step.

Also note that Parlay/OSA permits the authentication steps (1 and 4) to be skipped for entities within the same domain, for example, when an operator adds a new SCS to its Framework or if the application is in the same domain as the Framework and SCSs. It is a general misconception that Parlay/OSA is focusing only on third-party applications.

Physical Architecture

There can be numerous entities in the network offering the functionality behind the APIs: switches, interactive voice response systems (IVRs), HLRs, mobile positioning systems, Generic Packet Radio Service (GPRS) support nodes (SGSNs), billing servers, and so on. As mentioned, SCSs are logical entities, and the specifications do not prescribe whether SCSs should be separate boxes in the network. In principle, an SCS can be deployed as a standalone node in the network or directly on a node in the core network. For example, the user status SCS could be deployed directly on the HLR or on a separate standalone node that uses the CAMEL protocol [12] for communicating with the HLR. The two options are described in more detail in [16]. In the first option, the SCS is basically a component part of the network entity that communicates with applications via the API and addresses the functionality of the network entity via an internal interface. In the latter option, the SCS basically “talks API” with the applications and “protocol” with the core network. 3GPP has created a number of informative technical reports that show the detailed mapping between some of the APIs and network protocols [9].

Where the SCSs are deployed on separate standalone nodes, one can distinguish a physical service enabler sublayer in the network (Fig. 3). One solution here is to provide all API implementations within one physical node. Usually, this is referred to as the physical OSA gateway. This gateway has protocols/interfaces to all the various core network entities. The other approach is a distributed approach. In this case the OSA gateway node contains the Framework and maybe a few SCS components, but the rest of the SCSs run on different nodes.

All deployments are possible in principle, but the appropriate middleware infrastructure is in place between different nodes. However, there are cases where it is not desirable to deploy SCS software directly on a core network node. An example of such a node is the switch in the mobile network (MSC). Usually there are many such switches in a particular network, and a mobile subscriber can be attached to any of them, based on the location of the subscriber. When these nodes also serve as SCSs it means that all of them need to be able to set up communication with the application.

In practice there will be a mixture of all indicated deployment scenarios. There will most likely be a gateway node offering the

Parlay/OSA Framework that includes the registration interface and some core SCSs. Additional SCSs can then run on separate nodes, and be registered to the Framework in the gateway node. This also allows phased introduction of SCSs. Furthermore, as the Framework is a critical entity, it should be deployed on a node that provides the usual telecom carrier class characteristics like 99.999 percent availability.

Security Considerations

In order to allow applications from other domains to use the SCSs in the network, all communication between the application server and the SCSs must be secure.

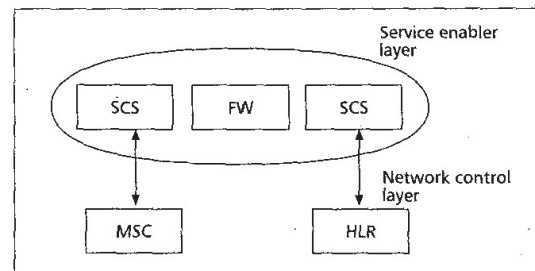
As mentioned earlier, Parlay/OSA includes authentication between domains. The mechanisms used rely on well-known authentication mechanisms and are also extendable with new ones. However, apart from authentication on the application level, one also has to ensure that the underlying communication is secure and that access to capabilities is controlled. These two topics will be addressed below.

Secure Communication — Secure communication can be realized with standard IT technology, firewalls, SSL, IPSec, and so on. It is also not expected that application servers will communicate directly over the Internet with SCSs, mainly for security and performance reasons.

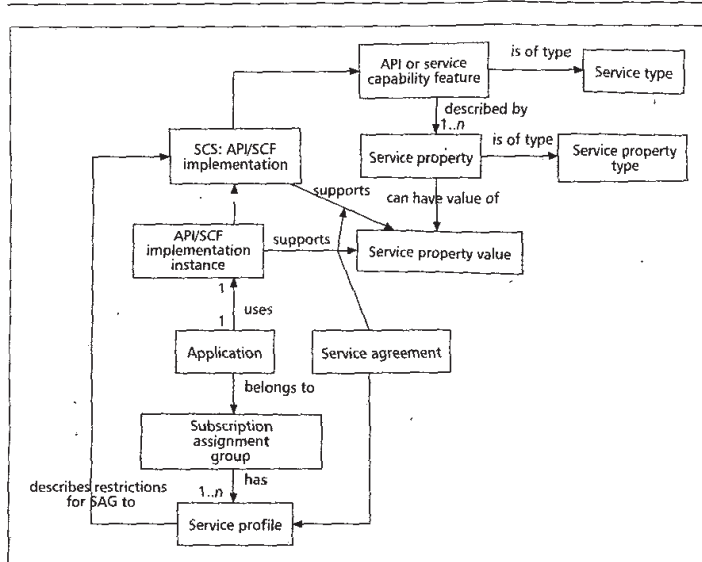
A more likely scenario is that dedicated connections are used. Thus, it should be ensured that only Parlay/OSA communication is allowed over the dedicated link, and also that the applications can be assured that nobody else is able to address the application peer objects present at the gateway. This can be achieved by introducing a firewall that only allows the specific communication protocol (e.g., CORBA) to pass, and using Secure Socket Layer (SSL) to prevent other applications from “stealing” object references.

Another security issue is how the integrity of a service agreement can be guaranteed. For this we have to focus on the service properties and service agreements.

Controlled Access: Service Agreements and Service Properties — Earlier we briefly stated that during registration an SCS publishes its capabilities. Based on these capabilities, the Framework operator is able to enter the conditions under which an application is allowed to use the SCS. This section focuses on access control and provides more details than an earlier section.



■ Figure 3. One specific network configuration where all SCSs are deployed on separate nodes. In this configuration one can distinguish a physical service enabler sublayer in the network architecture.



■ Figure 4. A model showing the concepts related to service agreements.

In Fig. 4 a UML model outlines a static view of all the concepts related to the service agreement, which captures what an application is and is not, allowed to do. This data view is helpful for understanding the different concepts and their relations. The upper part of the model is especially related to the registration and runtime communication establishment phases, while the lower part relates to the setting up of service agreement and runtime communication-establishment phases.

Starting at the top middle of the figure and focusing on the registration phase, the model shows that an API or SCF is of a certain type (e.g., call control, user interaction) and is described by a set of service properties. Service properties are of a certain type (e.g., string) and can have a certain value. General properties include the name of the API/SCF, version, name of the implementation, and version of the implementation. In addition, each type of API has its own properties; for instance, call control has a property to indicate the maximum number of parties in one call session. For a specific implementation of an API, the properties are assigned values (e.g., a value of 8 for the maximum number of parties in a call); this allows the SCS to publish its capabilities and restrictions during registration to the Framework.

At the bottom of the figure, the model shows the service profile concept used to specify which values (within the range of supported service property values) applications are allowed to use. For example, it might be that although the SCS supports up to 8 parties/call, the application is allowed to use only up to 4. When the application actually selects one of the implementations returned during the discovery phase, a service profile becomes active and is called a service agreement.

The final concept shown in the UML model is the subscription assignment group. The subscription assignment group contains service profiles for all the APIs needed by the applications within such a group. It is thus a collection of service profiles that are applicable to a set of applications.

Earlier it was shown how SCSs are registered to the Framework, how they can be discovered by applications, and how an instance is created when the application decides to use the SCS. At registration the Framework thus receives all the supported property values of the SCS. The Framework should then store these values in some kind of database. Each time a new application is introduced, it should be assigned to a subscription assignment group. After the discovery procedure, when the application

selects the SCS to use and signs the service agreement, the Framework propagates the values of the service properties specifying the capabilities that the application is allowed to use in communication with the SCS. The SCS then creates an instance that has a one-to-one relationship to the application; during runtime this instance should also check that the application does not violate the service agreement.

This clarifies the power of the service properties and service agreement when it comes to controlled access. One can easily indicate what applications are allowed to do, say, by creating the profile in such a way that the application is not allowed to use certain operations or alter certain parameters.

Quality of Service

Service agreements and policies can also be used for quality of service (QoS) aspects. Policies and service properties (e.g., guaranteeing a minimum number of sessions maximum number of allowed sessions per second) will soon be introduced.

Scalability

Another major nonfunctional architectural aspect is scalability. Parlay/OSA allows for scalability on various levels.

First of all, it is possible to add more SCSs and distribute the load from different applications over multiple SCSs. The way this works is that at the service selection phase, the Framework diverts one application to one SCS and another to a different SCS. Of course in the case where the specific SCS depends on network entities, this scalability is limited by the capacity of the core network.

Furthermore, with underlying middleware like CORBA it is possible to distribute load on a session basis without the application being aware that different sessions run on different processors.

Finally, in some of the APIs it is possible to add multiple application callbacks to the SCS, which allows the SCS to distribute the load of different sessions over different applications running on different servers. However, there is currently no means for applications to indicate a distribution policy.

Performance, Availability, Robustness, Redundancy

Being the interface between telecom and datacom networks, the SCS has to provide the usual telecom characteristics like 99.999 percent availability, ensuring that it is capable of handling a lot of parallel transactions from the application and network sides, without much latency and so on. There is no explicit support for all of these nonfunctional aspects in the Parlay/OSA APIs, as this is related to product implementation. However, the P1110 project from Eurescom has created a white paper that assesses the nonfunctional aspects of Parlay/OSA [13].

The main challenge to providing high-performance high-availability solutions will be to ensure that applications also provide these characteristics.

The Application Layer

Application Servers

Besides the SCSs, much is also expected for ease of application development from the application servers themselves. From the previous sections it is clear that some kind of distribution of SCSs is very likely; therefore, middleware like CORBA is needed. However, application servers allow functional distribution to be hidden and offer application developers a standard pro-

programming environment in their favorite language. Application servers are also the most suitable environments for use of the work of the JAIN community. JAIN focuses on the specification of the APIs in Java. At this moment most of the Parlay APIs are also specified in Java.

The functionality of the Parlay/OSA APIs may even be captured in development components (e.g., Java Beans) that allow application developers to link in and tune the required components. Service development kits (SDKs) are expected to further speed up the application development process. One crucial aspect of these development kits is that it should be easy to link in new components, offering communication with a new SCS that has been made available in the network.

Sample Parlay/OSA Applications

From the four main starting points listed earlier, one can distinguish different types of applications.

Since one of the starting points was to be able to access a broad range of functions in the network, Parlay/OSA allows applications to combine capabilities that were not easy to combine with technologies like IN. Call setups can easily be combined, by requesting the status of a user and then requesting a reservation for charging purposes. The first type of application one can distinguish is thus one that combines different capabilities. An example is an application the subscriber can use to compile a list of people (buddy list) and request to be notified when these people get in the neighborhood of the end user. Once a set of people from the list is available, the application can set up a multiparty call or send a short message.

Below, we have listed the sort of communication that may take place, along with the necessary interactions:

- The nontraffic part: The application contacts the Framework, is authenticated depending on whether it is in a different domain, and then requests for access to the user location/user status, call control, and user interaction APIs. The Framework will check that the application is allowed to use the requested APIs and, if so, will request the different SCSs to create an instance to be used by the application. The references of these instances will be returned to the application (step 1 in Fig. 5).
- An end user is now able to access the application (e.g., via the Internet or Wireless Access Protocol, WAP) to set up the buddy lists and related conditions for the notifications (step 2 in Fig. 5).
- When this list is complete, the application contacts the user location/user status SCS and requests to be notified when the people within the list are in a certain area (step 3 in Fig. 5).
- The SCS will initiate installation of triggers in the network in order to be notified when people from the buddy list get within a certain area (step 4 in Fig. 5).
- Each time a person from the list turns up in the area, the user location/user status SCS is informed (step 5 in Fig. 5).
- The user location/user status SCS notifies the application (step 6 in Fig. 5).
- If the application finds out that enough people are available, it informs the user, by either popping up a window, using the call control SCS to place a call with an announcement, or using the user interaction SCS to send an SMS message (step 7 in Fig. 5).

- The user decides to request the application to set up a conference call with all parties available (step 8 in Fig. 5).
- The application could check whether the end user still has enough credit by contacting the charging SCS (step 9 in Fig. 5).
- The application contacts the call control SCS to initiate the conference call (step 10 in Fig. 5).

By supporting multiple business models, Parlay/OSA also allows a combination of network capabilities with data not traditionally found in the telecom network. The second type of application is thus one that combines network capabilities with enterprise data. A typical example would be an application that can direct the end user to the location of the nearest automatic teller machine (e.g., when the user is in a city he/she has not been in before). The location of ATM machines is data not usually found in the traditional telecom network that also often "belongs" to the enterprise in question (the bank). Other examples that combine business data with network capabilities include applications that notify users of forecasts of extreme weather conditions or interesting sales opportunities when a user is in the neighborhood of a certain shop.

By enabling network convergence through the right level of abstraction of the APIs, it is also possible to deploy SCSs in the enterprise domain, such as in a private branch exchange (PBX), with the call control API. Enterprise applications can thus be located in the enterprise domain, and end users can then access these applications when they are within the corporate network. They can also access these applications even when they are outside the enterprise network (e.g., when traveling). To set this up would require an agreement between the application provider (enterprise) and the network operator. In this case the application is not part of the standard application portfolio of a certain operator, but only accessible to the group of corporate users. We will refer to these types of applications as corporate applications. Of course, it might also be that when present inside the corporate domain, the corporate application is accessed by other mechanisms than Parlay/OSA, but when outside the corporate environment, Parlay/OSA is used to access the application. A typical example would be the enterprise mail system, also accessible by dialing into an application that is able to send the mail messages to a text-to-speech engine via the Parlay/OSA APIs.

One special case here is the mission-critical application that uses some SCSs of the network operator and is used by only a

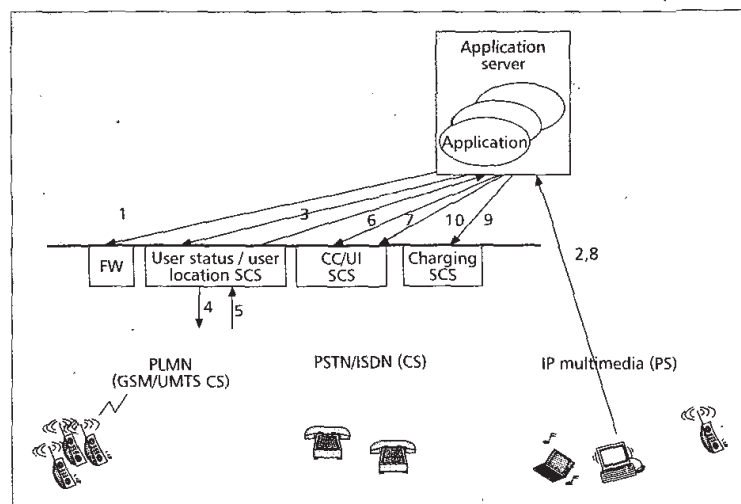


Figure 5. An application combining different capabilities.

very limited set of corporate users. A typical example would be an application that can inform a worker in the field closest to the location of a certain event. What is needed is an application able to access the user location API to track the mobile stations of the workers and also able to access, say, the call control API to place a call or the user interaction API to send an SMS message.

These are only a handful of examples; Parlay/OSA opens up such a range of possibilities that there are few bounds to the range of applications possible.

Future and Evolution of Parlay/OSA

The set of APIs developed within Parlay/OSA has grown substantially within a few years, and a lot has been achieved, as mentioned before. Under development is the further integration of Parlay/OSA with IP multimedia (e.g., the 3GPP all-IP-based core network [17]) and the possibility to link in with other SIP-based networks. At this moment the Joint API group is working on these aspects, and it is expected that some specific APIs like multimedia call control may need some minor evolution. One important aspect to consider is that Parlay/OSA and SIP-based approaches (e.g., SIP servlets), are not competing with each other, but are complementary. This is because Parlay/OSA provides access to a large range of network capabilities from existing 2G networks, a secure infrastructure to support third-party applications, and a higher-level abstraction to many capabilities, not just session control; and SIP provides a broad but lower-level access to multimedia sessions and is more comparable to IN protocols [15, 17].

Due to the quick progress, one improvement area is that specifications should be made more readable and simplified. A high-level description is needed, and the Parlay/OSA data types are especially complex in some cases. There is also the lack of a clear model that describes the relations of the different data types used in registration and discovery. However, this has been recognized and is under further development [14].

Work is also in progress in the area of policies. Currently the policy management API allows the definition and creation of policies. However, work is in progress to define per type of API, the policies that can be applied. This work might lead to further evolution of the definition of the service agreement and service properties.

Finally, Parlay is actively working on definitions of Webservice in order to allow the Webservice developer community to use services offered by the capabilities in the network. Currently, this work is undertaken by the Parlay X Working Group within Parlay, and will be aligned with the work of OMA that recently started.

It is important to realize that Webservices over additional means to open up the network, complementing the work that has finished. In offering Webservices technology, the networks would be opened up to an entirely new developer community. Applications that need to interact in real time with sessions in the network will use high-performance CORBA-based technology, but other applications could be implemented using either Java/CORBA or Webservice technology.

Conclusion

Parlay/OSA is becoming a powerful toolbox that provides the openness and flexibility for fast application development. By means of the joint API standardization working group, one common standard is achieved. The Parlay/OSA toolbox will also soon be extended with support for Web services.

Parlay/OSA is flexible when it comes to actual deployment in the network. For instance, it allows the introduction of new service capability, even ones that are not standardized. We have also shown there are several options to implement SCSs

and deploy them in the network. Furthermore, Parlay/OSA provides the necessary security for the core network to be opened up for third-party application providers.

We have outlined some typical Parlay/OSA applications. Today there are already SCS products and applications deployed that prove the Parlay/OSA concept and show the integration between telecom and datacom, without the end user being aware of this.

Acknowledgments

The authors gratefully acknowledge Erik van der Velden, Christophe Gourraud, Jörgen Dyst, and Chelo Abarca for input and helpful discussions. Furthermore, we would like to thank Musa Unmehopa, Andy Bennett, Paulus Karremans, Ultan Mulligan, and Richard Stretch for reviewing the article, and finally, Kimberleigh Anderson for a grammar review.

References

- [1] I. Faynberg et al., *The Intelligent Network Standards: Their Application to Services*, 1st ed., McGraw-Hill, 1996; or T. W. Abernethy and A. C. Munday, "Intelligent Networks, Standards and Services," *BT Tech. J.*, vol. 13, no. 2, Apr. 1995; <http://www.sc-server1.bt.com/btj/>
- [2] TINA Consortium, TINA Service Architecture v. 5.0, 1997; <http://www.tinac.com/> or H. Berndt et al., "TINA: Its Achievements and Its Future Directions," *IEEE Commun. Surveys and Tutorials*, vol. 3, no. 1, 1st qtr. 2000, <http://www.comsoc.org/surveys>
- [3] Parlay Group, <http://www.parlay.org/>
- [4] JAIN, <http://java.sun.com/products/jain/index.html>
- [5] J. de Keijzer et al., "JAIN: A New Approach to Services in Communication Networks," *IEEE Commun. Mag.*, Jan. 2000.
- [6] 3GPP, <http://www.3gpp.org/>
- [7] ETSI, <http://www.etsi.org/>
- [8] The Parlay Group, "Parlay Specifications 3.0," <http://www.parlay.org/specs/>
- [9] 3GPP "Open Service Access," Application Programming Interface," 3G TS 29.198 + TR 29.998; <http://www.3gpp.org/tftp/Specs/>
- [10] ETSI "Open Service Access," API, ES 201 915; <http://pda.etsi.org/pda/querysearch>
- [11] R.M. Stretch, "The OSA API and Other Related Issues," *BT Tech. J.*, vol. 19, no. 1, Jan. 2001; <http://www.sc-server1.bt.com/btj/>
- [12] 3GPP "Customized Applications for Mobile network Enhanced Logic," 3G TS 22.079, TS 23.078, TS29.078; <http://www.3gpp.org/tftp/Specs/>
- [13] G. Gylterud et al., White Paper on nonfunctional aspects and requirements related to Parlay/OSA products; <http://www.eurescom.de/public/projects/P1100-series/P1110/default.asp>
- [14] C. Moiso and S. Tognon, "Framework Information Model: A First Analysis," tech. doc. N5-020591, meeting CN5#19; obtainable via http://www.3gpp.org/tftp/tsg_cn/WG5_osa/
- [15] S. Desrochers et al., "Experimenting with PARLAY in a SIP Environment: Early Results," IP Telecom Svcs. Wksp. 2000, Atlanta, GA, Sept. 2000.
- [16] L. Bos and S. Leroy, "Towards an All-IP-UMTS-Based System Architecture," *IEEE Network*, Jan./Feb. 2001.
- [17] R. Pailer and J. Stadler, "A Service Framework for Carrier Grade Multimedia Services Using PARLAY APIs over a SIP System," *ACM Wireless Mobile Internet Wksp.*, Rome, Italy, July 2001.

Biographies

ARD-JAN MOERDIJK (Ard-Jan.Moerdijk@eln.ericsson.se) holds a Ph.D. in theoretical physics. After his Ph.D. work he moved to the IT and telecom industry. When he joined Ericsson he became involved in standardization, first in the Object Management Group and soon also in the early stages of Parlay. Next he became responsible for getting the Parlay APIs standardized in 3GPP in the context of OSA. In parallel he became involved in product management in this area and has been leading a standards team. Mid-2001 he was elected Chair of 3GPP CN5, the group responsible for the OSA APIs. Nowadays he is also responsible for all technology aspects of New Business and Innovations for Ericsson products in the IN area and leading a system management team.

LUCAS KLOSTERMANN (Lucas.Klostermann@eln.ericsson.se) holds a Ph.D. in physics which he obtained while working at CERN in Geneva on a large-scale high-energy physics experiment. After this period he made the switch to industry, starting with Ericsson in a research position in the area of IN. He worked on various projects related to distributed service environments, and got involved in different areas of standardization (IN, voice over IP, Parlay/OSA). He was Chair of the 3GPP CN5/OSA group during the work on 3GPP OSA R4, before handing over to Ard-Jan Moerdijk. In parallel to standardization work he has been product manager in the initial phases of Ericsson's Parlay/OSA Gateway. Presently he leads a small technology development/systems management group that works in close collaboration with customers and product development, with a focus on the further evolution of Ericsson's service architecture solutions.